CS340 FINAL
DATABASES

08/18/2017

BRIDGE INSPECTION DATABASE

By Kevin Lewis

Live Website:

http://flip1.engr.oregonstate.edu:2334/bridge

Bridge Inspection Database Outline

The Federal Highway Administration through the National Bridge Inspection (NBI) program requires that a safety inspection is performed for all bridges once every two years at a minimum. This is to ensure that bridges are maintained such that they are safe to use by the public. In practice. This entails documenting any deficiencies in the bridge, example a crack in a beam, so that the owners, such as the state government, can identify what maintenance will needs to be performed.

The inspection process is imperative for what is known as Fracture Critical bridges. These are typically bridges with two or less load carrying members such that if one of the members fails, the entire bridge will collapse. An excellent example is a through truss bridge, see photo below. The top and bottom horizontal members are called the top and bottom chord. If either of those fails on one side of the bridge, then the entire bridge will fail. Because Fracture Critical bridges are prone to sudden failure, they are required to be inspected once a year.

Bridge inspection project management is a complex task that involves juggling personnel and equipment to ensure that bridges are inspected on-time and on-budget while producing a high-quality deliverable for the client. Currently, at my company it is difficult for a project manager to take a snap-shot of the project at any point. He or she must rely on employee timesheet input that eventually percolates up to the project manager, who then needs to parse the information

to determine the status of a bridge. However, bridge inspection work can sometimes take place over several weeks, and can give a false indication that a bridge is below budget. The purpose of the database is to give more tools to the project manager to better track the status of a bridge's inspection.

Database Outline

The following is a detailed description of the various elements within the Bridge Inspection Database.

Employee:

- Description: This entity are the people at the company that will take part in managing the program, inspecting the bridges, and writing the reports.
- Properties:
    - Employee ID Primary Key
    - First Name
    - Last Name
    - Hourly Rate

- Relationship:
    - Assigned to an inspection (many-to-many): Each employee can be assigned to an inspection. An inspection means the person is either planning or going out to the bridge. Within the inspection assignment, the employee will have a role, such as team leader, inspector, inspection manager. Employees are not required to be assigned to any inspections.
    - Assigned to a report (many-to-many): Like the inspection assignment, each employee can be work on a report associated with a report associated with an inspection. Each employee will be associated with a task within the relationship
    - Has-an Employee ID (one-to-one): Each employee has one employee ID

Office:

- Description: Each office location where the employee works.
- Properties:
    - ID (Primary Key)
    - Name
    - Zip Code

- Relationship:
    - Has employees (one-to-Many): Has at least one employee who works there

Bridge:

- Description: Each bridge to be inspected.
- Properties:
    - Bridge ID
    - Type
    - Span
    - Length
    - Zipcode
- Relationship:
    - Has a report (one-to-many): Each bridge has one to many reports. Typically it will only be one, but there could be different inspection (special vs routine) which would require two separate inspections.
    - Has an inspection (one-to-many): Like the reports, each bridge has one to many inspections.
    - Is associated with an LOA (Many to One): Each bridge is only associated with one LOA

Report:

- Description: The report that are associated with the bridges and typically paired with an inspection.
- Properties:
    - ID
    - Type : Special vs Routine, etc.
    - Budget
    - Status (text)
    - Percent complete
- Relationship:
    - Has a bridge/LOA (many-to-one): Each report has only one bridge and one LOA.
    - Many employees work on it (one to Many)

Inspection:

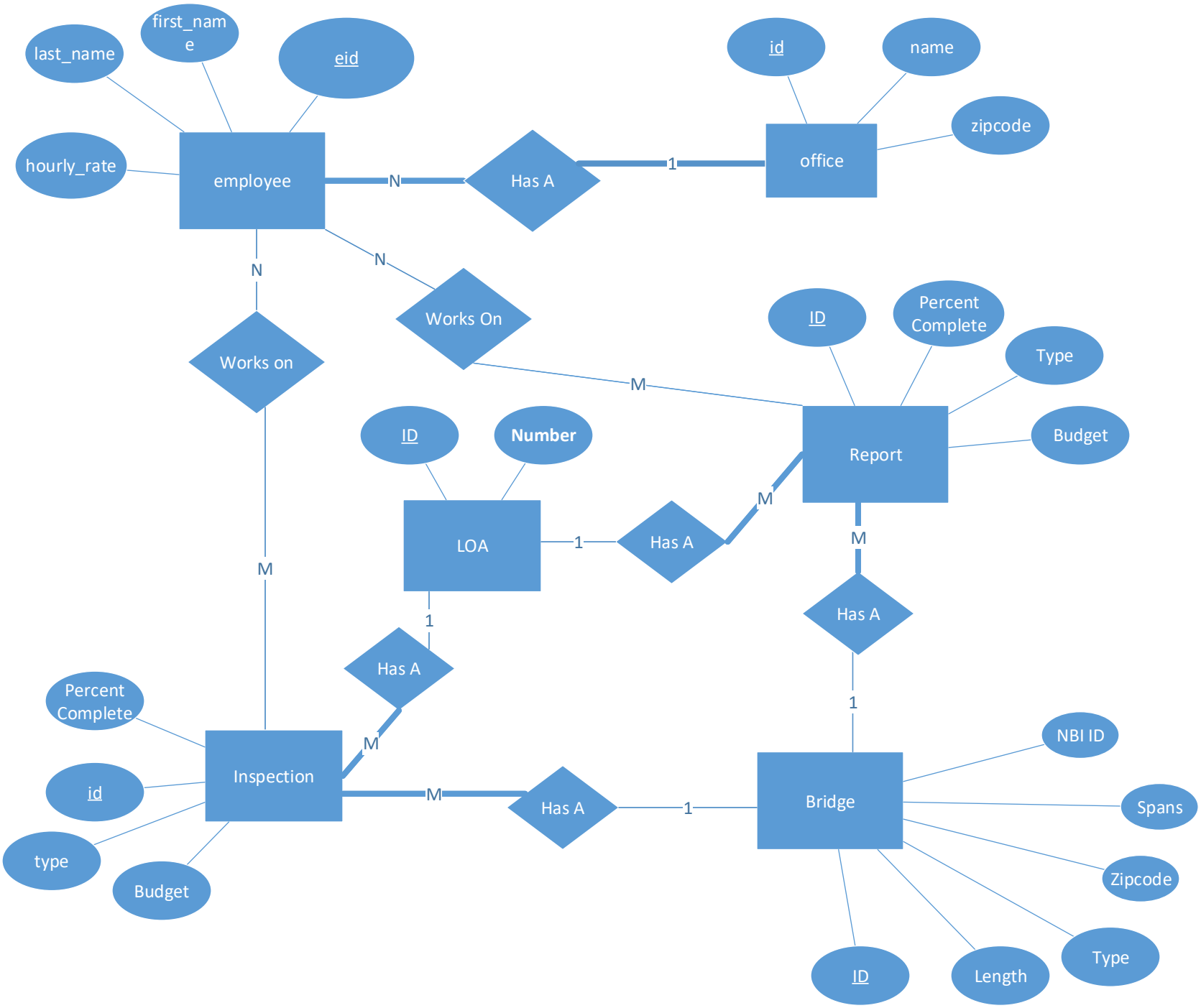
- Description: Inspection associated with the bridge
- Properties:
  - ID (Primary Key)
  - Type: Special vs Routine, etc.
  - Budget
  - Status (text)
  - Percent complete
- Relationship:
  - Has a bridge (many-to-one): Each inspection has only one bridge.
  - Many employees work on it (one to Many)

LOA (Letter of Authorization):

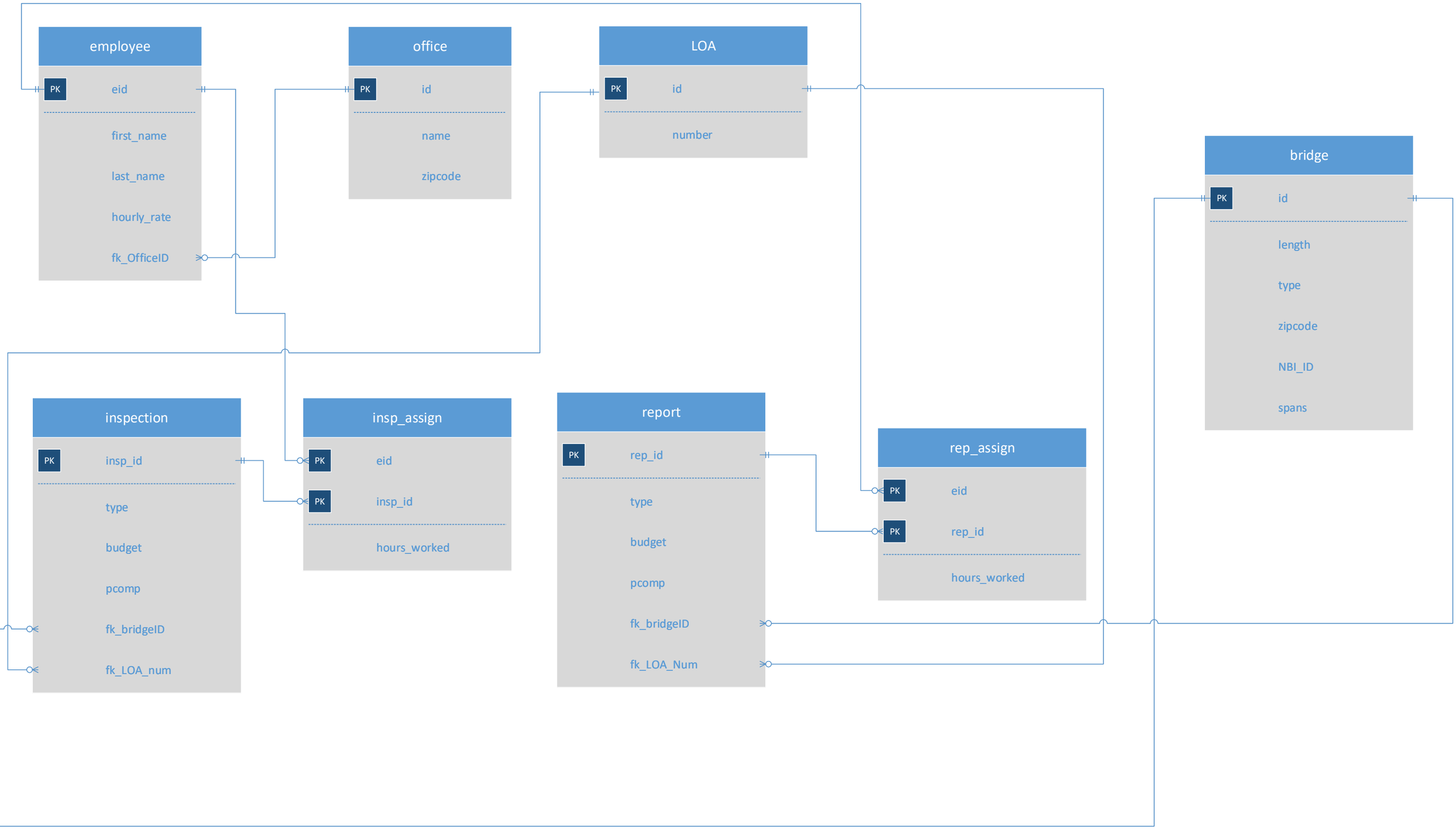
- Description: The LOA is the overall section of the project that the bridges are associated with. It is at this level that Project Managers mostly see if a project is over budget or not.
- Properties:
  - ID
  - Number
- Relationship:
  - Has at least one report / inspection (one-to-many): The loa has several reports or inspections, but each inspection / report must have one loa.

ER Diagram

ER Diagram


last_name
first_name
eid
hourly_rate
employee
N
Has A
1
id
name
zipcode
office
N
N
Works on
Works On
M
ID
Percent Complete
Type
Budget
Report
ID
Number
LOA
1
Has A
M
M
Has A
1
Has A
M
Percent Complete
id
type
Budget
Inspection
M
Has A
1
Bridge
NBI ID
Spans
Zipcode
Type
Length
ID

Schema


employee
PK
eid
first_name
last_name
hourly_rate
fk_OfficeID
office
PK
id
name
zipcode
LOA
PK
id
number
bridge
PK
id
length
type
zipcode
NBI_ID
spans
inspection
PK
insp_id
type
budget
pcomp
fk_bridgeID
fk_LOA_num
insp_assign
PK
eid
PK
insp_id
hours_worked
report
PK
rep_id
type
budget
pcomp
fk_bridgeID
fk_LOA_Num
rep_assign
PK
eid
PK
rep_id
hours_worked

Data Definition Queries

```
CREATE TABLE employee (
    eid INT AUTO_INCREMENT NOT NULL,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    hourly_rate FLOAT,
    fk_officeID INT,
    PRIMARY KEY(eid),
    CONSTRAINT fl_employee UNIQUE (first_name, last_name),
    FOREIGN KEY (fk_officeID) REFERENCES office(id)
) ENGINE=InnoDB


CREATE TABLE office (
    id INT AUTO_INCREMENT NOT NULL,
    name VARCHAR(255) NOT NULL,
    zipcode INT,
    PRIMARY KEY(id),
    UNIQUE (name)
) ENGINE=InnoDB

CREATE TABLE bridge (
    id  INT AUTO_INCREMENT NOT NULL,
    length INT,
    type varchar(255),
    zipcode INT NOT NULL,
    spans INT,
    NBI_ID INT NOT NULL,
    PRIMARY KEY (id),
    UNIQUE (NBI_ID)
) ENGINE=InnoDB

CREATE TABLE LOA (
    id INT AUTO_INCREMENT NOT NULL,
    number INT NOT NULL,
    PRIMARY KEY (id),
    UNIQUE (number)
) ENGINE=InnoDB
```

```
CREATE TABLE inspection (
    insp_id INT AUTO_INCREMENT NOT NULL,
    type varchar(255),
    budget FLOAT NOT NULL,
    pcomp INT,
    fk_bridgeID INT NOT NULL,
    fk_LOA_num INT NOT NULL,
    PRIMARY KEY (insp_id),
    FOREIGN KEY (fk_bridgeID) REFERENCES bridge(id) ON DELETE CASCADE,
    FOREIGN KEY (fk_LOA_num) REFERENCES LOA(id)
) ENGINE=InnoDB

CREATE TABLE report (
    rep_id INT AUTO_INCREMENT NOT NULL,
    type varchar(255),
    budget FLOAT NOT NULL,
    pcomp INT,
    fk_bridgeID INT NOT NULL,
    fk_LOA_num INT NOT NULL,
    PRIMARY KEY (rep_id),
    FOREIGN KEY (fk_bridgeID) REFERENCES bridge(id) ON DELETE CASCADE,
    FOREIGN KEY (fk_LOA_num) REFERENCES LOA(id)
) ENGINE=InnoDB

CREATE TABLE insp_assign(
    eid int NOT NULL,
    insp_id int NOT NULL,
    hours_worked int,
    PRIMARY KEY (eid, insp_id),
    FOREIGN KEY (eid) REFERENCES employee(eid),
    FOREIGN KEY (insp_id) REFERENCES inspection(insp_id)
) ENGINE=InnoDB

CREATE TABLE rep_assign(
    eid int NOT NULL,
    rep_id int NOT NULL,
    hours_worked int,
    PRIMARY KEY (eid, rep_id),
    FOREIGN KEY (eid) REFERENCES employee(eid),
    FOREIGN KEY (rep_id) REFERENCES report(rep_id)
) ENGINE=InnoDB
```

Data Manipulation Queries

SQL = BLUE
Variables = RED

Report Status Site:

```
SELECT s_rep.rep_id AS elem_id, s_rep.NBI_ID, s_rep.bridgeID, l.number,
IFNULL(s_rep.total_rep,0) AS total_spent, IFNULL(s_rep.budget,0) AS
total_budget, IF( IFNULL(s_rep.total_rep,0) - IFNULL(s_rep.budget,0) < 0,
'Over Budget', 'Under Budget') AS cur_status FROM LOA l
INNER JOIN (SELECT r.rep_id, b.NBI_ID, b.id AS bridgeID, r.fk_LOA_num as
                    LOA_num, SUM(e.hourly_rate * ra.hours_worked) AS total_rep,
                    SUM(r.budget) as budget FROM report r
                   LEFT JOIN rep_assign ra ON ra.rep_id = r.rep_id
                   LEFT JOIN employee e ON e.eid = ra.eid
                   INNER JOIN bridge b ON b.id = r.fk_bridgeID GROUP BY r.rep_id)
AS s_rep ON s_rep.LOA_num = l.number
GROUP BY elem_id [HAVING cur_status = "Over Budget", HAVING cur_status =
"Under Budget", HAVING l.number = ?, HAVING bridgeID = ?, HAVING elem_id
= ?]
```

Query Comments: The above queries overall goal is to determine if an reoprt is over-budget. It starts by pulling the information that we need from the report by joining with the various tables. The larger inner selection that it joins with determines the total amount spent on each project by summing the hours x hourly wage for each employee working on the report.

The user is given several options of filtering the data (see the final group by statement) including the status, loa number, bridge NBI and report number.

The following is used to fill out tables on the webpage:

```
SELECT l.id, l.number FROM LOA l
INNER JOIN inspection i ON i.fk_LOA_num = l.id

SELECT DISTINCT b.id, b.NBI_ID FROM bridge b
INNER JOIN inspection i ON i.fk_bridgeID = b.id

SELECT rep_id AS elem_id FROM report
```

Inspection Status Site:

```
SELECT tmp.insp_id AS elem_id, tmp.NBI_ID, tmp.bridgeID, l.number,
IFNULL(tmp.total_insp,0) AS total_spent, IFNULL(tmp.budget,0) AS
total_budget, IF(IFNULL(tmp.total_insp,0) - IFNULL(tmp.budget,0) < 0, 'Over
Budget', 'Under Budget') AS cur_status FROM LOA l
INNER JOIN (SELECT i.insp_id, b.NBI_ID, b.id AS bridgeID, i.fk_LOA_num as
LOA_num, SUM(e.hourly_rate * ia.hours_worked) AS total_insp,
                    SUM(i.budget) as budget FROM inspection i
                    LEFT JOIN insp_assign ia ON ia.insp_id = i.insp_id
                    LEFT JOIN employee e ON e.eid = ia.eid INNER JOIN bridge b
                    ON b.id = i.fk_bridgeID GROUP BY i.insp_id)
AS tmp ON tmp.LOA_num = l.number GROUP BY elem_id [HAVING cur_status
= "Over Budget", HAVING cur_status = "Under Budget", HAVING l.number = ?,
HAVING bridgeID = ?, HAVING elem_id = ?]
```

Query Comments: Following a similar structure as the report, the status of the inspections are determined.

The user is given several options of filtering the data (see the final group by statement) including the status, loa number, bridge NBI and report number.

The following is used to fill out tables on the webpage:

```
SELECT l.id, l.number FROM LOA l
INNER JOIN inspection i ON i.fk_LOA_num = l.id

SELECT DISTINCT b.id, b.NBI_ID FROM bridge b
INNER JOIN inspection i ON i.fk_bridgeID = b.id

SELECT insp_id AS elem_id FROM inspection
```

LOA Status Site:

```
SELECT l.number, IFNULL(s_insp.total_insp,0) + IFNULL(s_rep.total_rep,0) AS
total_spent, IFNULL(s_insp.budget,0) + IFNULL(s_rep.budget,0) AS
total_budget, IF((IFNULL(s_insp.total_insp,0) + IFNULL(s_rep.total_rep,0)) -
IFNULL(s_insp.budget,0) + IFNULL(s_rep.budget,0) < 0, 'Over Budget', 'Under
Budget') AS cur_status FROM LOA l
LEFT JOIN (SELECT i.insp_id, i.fk_LOA_num as LOA_num,
            SUM(e.hourly_rate * ia.hours_worked) AS total_insp, SUM(i.budget)
            as budget FROM inspection i
            INNER JOIN insp_assign ia ON ia.insp_id = i.insp_id
            INNER JOIN employee e ON e.eid = ia.eid GROUP BY i.insp_id)
            AS s_insp ON s_insp.LOA_num = l.number
LEFT JOIN (SELECT r.rep_id, r.fk_LOA_num as LOA_num, SUM(e.hourly_rate *
            ra.hours_worked) AS total_rep, SUM(r.budget) as budget FROM
            report r
            INNER JOIN rep_assign ra ON ra.rep_id = r.rep_id
            INNER JOIN employee e ON e.eid = ra.eid GROUP BY r.rep_id)
AS s_rep ON s_rep.LOA_num = l.number GROUP BY l.number  [HAVING
cur_status = "Over Budget", HAVING cur_status = "Under Budget", HAVING
l.number = ?]
```

Query Comments: See the comments from the Inspection and Report statuses. This will combine both the total spent and the total budget on the inspections and reports. The totals are compared to determine if an loa is over or under budget.

The user is given several options of filtering the data (see the final group by statement) including the status, loa number, bridge NBI and report number.

The following is used to fill out tables on the webpage:

```
SELECT * FROM LOA
```

Add Employee Site:

Data Manipulation

```
UPDATE employee SET hourly_rate = [?] WHERE eid = [?]

INSERT INTO employee (eid, first_name, last_name, hourly_rate, fk_OfficeID)
VALUES (?, ?, ?, ?, ?)
```

Gets data for the site

```
SELECT e.eid, e.first_name, e.last_name, e.hourly_rate, o.name AS office_name
FROM employee e
INNER JOIN office o ON e.fk_Officeid = o.id ORDER BY e.eid

SELECT * FROM office
```

Add Report Site:

Data Manipulation

```
INSERT INTO report (type, budget, pcomp, fk_bridgeID, fk_LOA_num) VALUES
(?, ?, ?, ?, ?)
```

Gets data for the site

```
SELECT r.rep_id, r.type, r.budget, r.pcomp, LOA.number, b.NBI_ID FROM report
r
INNER JOIN LOA ON r.fk_LOA_num = LOA.id
INNER JOIN bridge b ON b.id = r.fk_bridgeID ORDER BY LOA.number

SELECT * FROM LOA ORDER BY number

SELECT * FROM bridge ORDER BY NBI_ID
```

Add Inspection Site:

Data Manipulation

```
INSERT INTO inspection (type, budget, pcomp, fk_bridgeID, fk_LOA_num)
VALUES (?, ?, ?, ?, ?)
```

Gets data for the site

```
SELECT i.insp_id, i.type, i.budget, i.pcomp, LOA.number, b.NBI_ID FROM
inspection i
INNER JOIN LOA ON i.fk_LOA_num = LOA.id
INNER JOIN bridge b ON b.id = i.fk_bridgeID ORDER BY LOA.number

SELECT * FROM LOA ORDER BY number

SELECT * FROM bridge ORDER BY NBI_ID
```

Add LOA Site:

Data Manipulation

```
INSERT INTO LOA (number) VALUES (?)
```

Gets data for the site

```
SELECT * FROM LOA ORDER BY number
```

Add Bridge Site:

Data Manipulation

```
INSERT INTO bridge (nbi_id, type, length, spans, zipcode) VALUES (?, ?, ?, ?, ?)
```

Gets data for the site

```
SELECT * FROM bridge ORDER BY NBI_ID
```

Add Office Site:

Data Manipulation

```
INSERT INTO office (name, zipcode) VALUES (?, ?)
```

Gets data for the site

```
SELECT * FROM office ORDER BY name
```

Assign Report Site:

Data Manipulation

```
UPDATE rep_assign SET hours_worked = ? WHERE eid = ? AND rep_id = ?
```

```
INSERT INTO rep_assign (eid, rep_id, hours_worked) VALUES (?, ?, ?)
```

```
DELETE FROM rep_assign WHERE eid = ? AND rep_id = ?
```

Gets data for the site

```
SELECT e.eid, e.first_name, e.last_name, e.hourly_rate, o.name AS office_name
FROM employee e
INNER JOIN office o ON e.fk_Officeid = o.id ORDER BY e.eid
```

```
SELECT r.rep_id, r.type, r.budget, r.pcomp, LOA.number, b.NBI_ID FROM report
r
INNER JOIN LOA ON r.fk_LOA_num = LOA.id
INNER JOIN bridge b ON b.id = r.fk_bridgeID ORDER BY r.rep_id"
```

```
SELECT e.eid, e.first_name, e.last_name, r.rep_id, r.type, b.NBI_ID, l.number,
ra.hours_worked FROM rep_assign ra
INNER JOIN employee e ON e.eid = ra.eid
INNER JOIN report r ON r.rep_id = ra.rep_id
INNER JOIN bridge b ON b.id = r.fk_bridgeID
INNER JOIN LOA l ON l.id = r.fk_LOA_num ORDER BY e.eid
```

Assign Inspection Site:

Data Manipulation

```
UPDATE insp_assign SET hours_worked = ? WHERE eid = ? AND insp_id = ?

INSERT INTO insp_assign (eid, insp_id, hours_worked) VALUES (?, ?, ?)

DELETE FROM insp_assign WHERE eid = ? AND insp_id = ?
```

Gets data for the site

```
SELECT e.eid, e.first_name, e.last_name, e.hourly_rate, o.name AS office_name
FROM employee e
INNER JOIN office o ON e.fk_Officeid = o.id ORDER BY e.eid

SELECT i.insp_id, i.type, i.budget, i.pcomp, LOA.number, b.NBI_ID FROM
inspection i
INNER JOIN LOA ON i.fk_LOA_num = LOA.id
INNER JOIN bridge b ON b.id = i.fk_bridgeID ORDER BY i.insp_id

SELECT e.eid, e.first_name, e.last_name, i.insp_id, i.type, b.NBI_ID, l.number,
ia.hours_worked FROM insp_assign ia
INNER JOIN employee e ON e.eid = ia.eid
INNER JOIN inspection i ON i.insp_id = ia.insp_id
INNER JOIN bridge b ON b.id = i.fk_bridgeID
INNER JOIN LOA l ON l.id = i.fk_LOA_num ORDER BY e.eid"
```